UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/754,017	01/07/2004	Kenneth W. Cowan	NUM-02402	7115

26339 7590 07/19/2007

MUIRHEAD AND SATURNELLI, LLC
200 FRIBERG PARKWAY, SUITE 1001
WESTBOROUGH, MA 01581

EXAMINER	
VU, TUAN A	

ART UNIT	PAPER NUMBER
2193	

MAIL DATE	DELIVERY MODE
07/19/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/754,017	COWAN ET AL.	
	Examiner	Art Unit	
	Tuan A. Vu	2193	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *23 April 2007*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1,3,4,6-17,19,20,22,24,25,27-38,40 and 41* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1,3,4,6-17,19,20,22,24,25,27-38,40 and 41* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This action is responsive to the Applicant's response filed 4/23/07.

As indicated in Applicant's response, claims 1, 3-4, 6, 8, 11-12, 16-17, 19-22, 24-25, 27-38, 40-41 have been amended, and claims 2, 5, 18, 21, 23, 26, 39, 42 canceled. Claims 1, 3-4, 6-17, 19-20, 22, 24-25, 27-38, 40-41 are pending in the office action.

Claim Objections

2. Claim 19 is objected to because of the following informalities: the dependency to "Claim 19" is a typographical informality, and should be treated as 'claim 16'. Appropriate correction is required.

Claim Rejections - 35 USC § 102

3. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

4. Claims 1, 3-4, 6-10, 16-17, 19-20, 22, 24-25, 27-31, 37-38, 40-41 are rejected under 35 U.S.C. 102(b) as being anticipated by Leblang et al., USPN: 5,574,898 (hereinafter Leblang).

As per claim 1, Leblang discloses a computer-implemented method for automatically tracking build information comprising:

extracting build information by processing, for each of (derived objects, configuration record – col. 25, lines 27-55) one or more builds one or more software modules produced using a compilation (see *foo.c, foo.h, foo.o* - Fig. 23; Fig. 20) process resulting in said one or more modules; wherein said build information extracted includes software module information about

at least one software module produced as an output of a compilation process for each of said one or more builds (e.g. Fig. 20-23; see *foo.c, foo.h, foo.o* - Fig. 23);

registering said one or more builds by storing said build information (e.g. VOB – Figs. 6-7; repository 102, VOB – Fig. 24) corresponding to each of said one or more builds in a database;

automatically determining software module information about software being tested (e.g. col. 9, lines 10-55; *fixing a bug, test it* – col. 9, lines 40-66 – Note: check-out – see Fig. 9, 14 -- from version control system to check version delta – see Fig. 12-13 -- for developing code reads on determine runtime module information for module being tested); wherein said software module information is gathered (Fig. 22-24) from one or more software modules during execution of said software being tested (Note: a audit script for a development process reads on modules checked out and being tested for proper build determination – see Fig. 23; col. 9, lines 10-55); and

automatically determining a first of said one or more builds included in the database which corresponds to said software module information (e.g. col. 10, lines 44 to col. 11, line 57; Fig. 7; Fig. 22-24; col. 13, line 41 to col. 14, line 47 – Note: using configuration rules – see col. 9, lines 10-55 – to check appropriate version of components to be enlisted for a project of fixing a bug reads on determining a build that correspond to the obtained module information from the enlistment of project derived files or objects in view of col. 9, lines 40-66).

As per claim 3, Leblang discloses wherein said database that includes said build information is an object database (VOB – col. 9, lines 40-66; Fig. 24), and creating and storing one or more objects corresponding to each of said builds; and creating and storing one or more

objects corresponding to software modules included in each of the builds (Fig. 17, 20-21; 23, 24).

As per claim 4, Leblang discloses including:

creating and storing a session object (e.g. view: alpha –Fig. 21, 22; view … dynamic, can be modified - col. 9, lines 25-44 – Note: host client machine opening one instance of view reads on session object – see col. 12, lines 12-15, 49-53) corresponding to a test session (a check out of components for a release as to fixing a bug reads on test session – see col. 9, line 45 to col. 10, line 24) of said software being tested (*test it* – col. 9, lines 40-66; audit script – Fig 23);

creating and storing one or more objects (Fig. 20-21; 23-24; col. 10, lines 43-51) corresponding to software modules describing said software module information;

automatically determining a previously created build object (VOB- Fig. 23-24; VOB – Fig. 6) corresponding to one of said one or more builds previously registered; and storing an address of said previously created build object in said session object (e.g. entries *532* , record *102* – Fig. 23; Fig. 17; Fig. 8; */usr/hw/src/msg.o@@3-May.08:12* – Fig. 22).

As per claim 6, Leblang discloses gathering runtime information (e.g. col. 15 line 64 to col. 17, line 61 – Note: developer retrieving objects based on metadata during enlistment of persisted build components reads on gathering at runtime of development session – see Fig. 22-24) about software modules dynamically loaded (Note: view per developer reads on dynamic loading of code files – see Fig. 6; col. 9, line 8 to col. 10, line 35) in a computer system upon which said software being tested is executing.

As per claim 7, Leblang discloses subroutine calls (e.g. *checkout, checkin* - col. 18, lines 29-33; *Operating System/File system* 552 – Fig. 23) associated with an operating system executing in said computer system are used in said gathering runtime information.

As per claim 8, Leblang discloses using said build information in said database to automatically determine a second of said one or more builds (e.g. Fig. 17; Fig. 23) corresponding to software module information included in a bug report (e.g. *fixing a bug, test it* – col. 9, lines 40-66); and

creating and storing a bug report object corresponding to said bug report (Note: VOB record of derived object per access from a view for developing a release – see col. 13, line 41 to col. 14, line 47-- addressing a bug reads on bug report being store), said bug report object including an address associated with said second build (entries 532 - Fig. 22; File system -Fig. 23-24).

As per claim 9, Leblang discloses submitting said bug report included in a formatted electronic message (col. 13, lines 17-29 – NOTE: a file system call reads on electronic message); interpreting data included in said formatted electronic message (e.g. *checkout, checkin* - col. 18, lines 29-33; *Operating System/File system* 552 – Fig. 23) to enable said determining of said second build.

As per claim 10, Leblang discloses

creating and storing another build object corresponding to a build in which said bug report is identified as being corrected; and associating said other build object with said bug report object in said database (Fig. 23-24 – Note: release and VOB of previous builds reads on bug report of previous fix being persisted for reuse).

As per claim 16, Leblang discloses a computer-implemented method for determining a code volatility metric, the method comprising:

extracting build information by processing, for at least two builds, one or more software modules (e.g. derived objects, configuration record – col. 25, lines 27-55) produced using a compilation process (see *foo.c, foo.h, foo.o* - Fig. 23; Fig. 20) resulting in said one or more software modules; wherein said build information extracted includes software module information about at least one software module produced as an output of a compilation process for each of said one or more builds (e.g. Fig 23);

registering said at least two builds by storing said build information corresponding to each of said at least two builds (e.g. repository 102, VOB – Fig. 24) in a database;

identifying, by retrieving at least a portion of said build information from the database, a first and a second of said at least two builds (e.g. Fig. 22-23); performing a query of the database to determine code volatility between software modules included in both the first and the second of said at least two builds (Fig. 1, 6-7; Fig. 5 – Note: check out versions of modules to determine code changes reads on determining volatility between 2 builds based on build information in form of files checked out from comparison and eventual merging); and

calculating in response to said query, said code volatility metric using said build information including software module information about said first and said second builds included in the database, said code volatility metric being determined using one or more metrics (e.g. ΔB, ΔA, ΔC -- Fig. 12-13, 15-16) representing an amount of code change that has occurred between software modules in both said first build and said second build (Note: checking out versions of modules to determine code changes reads on determining volatility

between 2 builds based on build information in form of files being checked out from comparison and eventual merging).

As per claim 17, Leblang discloses determining a total number of functions of said first build and said second build; determining a percentage of functions added in accordance with said total number of functions; determining a percentage of functions removed in accordance with said total number of functions (e.g. Fig. 12-13, 15-16 – Note: code being removed, added in *delta* record reads on a relative portion – or percent of source code functions – of the ancestor version as set forth in the tree of stored versions) ;

determining a percentage of functions modified in accordance with said total number of functions; and

determining said code volatility metric as a sum of said percentage of functions added, said percentage of functions removed, and said percentage of functions modified (Note: sum of percent or portion of code being recorded as delta reads on code volatility computed as sum of source code functions changes).

As per claim 19, Leblang discloses determining differences in a function in which a first version of a function is associated with one software module and a second version of the function is associated with a second software module, said first software module being associated with a first build, and said second software module being associated with a second build (e.g. Clearmake col. 15, lines 37 col. 16, line 16); and using checksum or function signature information (e.g. *checksum* - col. 31, lines 5-38) for determining differences.

As per claim 20, Leblang discloses a method for tracking build information comprising:

extracting build information by processing, for at least two builds, one or more software modules (e.g. derived objects, configuration record – col. 25, lines 27-55 – Note: consulting a database of versioned objects reads on checking versioned objects previously destined for one or more builds – see *System Builder 550, VOB 102* - Fig. 24) produced using a compilation process (see *foo.c, foo.h, foo.o* - Fig. 23; Fig. 20) resulting in said one or more software modules; wherein said build information extracted includes software module information about at least one software module produced as an output of a compilation process for each of said one or more builds (e.g. derived object 500: ID, ref count, contents, config rec – Fig. 22; config record 102 – Fig. 23);

registering said one or more builds by storing said build information in a database (e.g. repository 102, VOB – Fig. 24);

executing a software program (e.g. *fixing a bug, test it* – col. 9, lines 40-66) that includes one or more software modules, said executing including processing (e.g. *invoking script, auditshell* – Fig. 23) said one or more software modules of the software program during execution of said program (Note: testing process, or bug fixing process including execution and checking code for audit purposes reads on program execution including processing modules for said execution – see col. 9, lines 40-66; Fig 23) to determine module information about said one or more software modules of said software program (e.g. Fig. 22-24);

automatically determining, using said build information included in the database and said module information obtained from said executing, a matching build (e.g. col. 10, lines 44 to col. 11, line 57; Fig. 7; Fig. 22-24 – Note: derived objects and audit information combined in a development process - *fixing a bug, test it* – col. 9, lines 40-66; Fig. 7 – reads on previous build

information in DB and module information from executing a program) for said one or more software modules that are also associated with one of said builds previously registered; and

determining testing information associated with the matching build (Note: audit execution reads on determining matching builds after comparing proper versions and corresponding discrepancies – see Fig. 6, 23) by performing a query of said database (e.g. *audit entries* – Fig. 23 – Note: any retrieval from a VOB reads on query being performed) said testing information including at least one type of runtime analysis performed for the matching build (Note: generating a audit record reads on performing a query from database of configuration previously stored).

As per claim 22, Leblang discloses a computer readable medium comprising machine executable code stored thereon for automatically tracking build information, the computer readable medium comprising machine executable for:

extracting build information by processing (… output of a compilation process for each of said one or more builds);

registering said one or more builds (… in a database);

automatically determining software module information (… gathered from one or more software modules … of said about software being tested); and

for determining a first of said one or more builds (… corresponding to said software module information);

all of which action steps having been addressed correspondingly in claim 1.

As per claims 24-25, 29-31, these claims correspond to the subject matter of claims 3-4, 8-10, respectively; hence will be rejected according to the rationale set forth therein, respectively.

As per claims 27-28, these claims correspond to the subject matter of claims 6-7, respectively; hence will be rejected according to the rationale set forth therein, respectively.

As per claims 37-38, 40, these claims correspond to the subject matter of claims 16-17, 19, respectively; hence will be rejected according to the rationale set forth therein, respectively.

As per claim 41, Leblang discloses a computer program product for tracking build information comprising machine executable code for:

extracting build information by ... ; registering said one or more builds by ... ; executing a software program that ... ; automatically determining, using said build information ... ; determining testing information associated... ;

all of which step limitations having been addressed in claim 20.

Claim Rejections - 35 USC § 103

5. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

6. Claims 11-15, 32-36 are rejected under 35 U.S.C. 103(a) as being un-patentable under Leblang et al., USPN: 5,574,898.

As per claim 11, Leblang discloses wherein said automatically determining said first build further comprises:

determining said first build for which a matching build is being determined from said one or more builds registered; determining a candidate list including one or more builds having at least one software module in common with said first build (e.g. Fig. 23-24; col. 13, line 41 to col. 14, line 47); for each build included in said candidate list, determining if modules included in said each build match software modules included in said first build (e.g. verify versions Fig. 24; col. 20, line 45 to col. 21, lines 63 – Note: each set of checkout files for a build reads on a candidate list);

But Leblang does not explicitly disclose

for each build included in said candidate list, if there are no software modules included in said each build that have a module name and associated attributes matching a software module included in said first build, determining that said each build is not a match for said first build; and

for each build included in said candidate list, if a first of said software modules included in said

each build has a software module name that matches a software module included in said first build but attributes associated with said software module do not match said software module included in said first build, determining that said each build is not a match for said first build.

Based on name and attributes associated with a record of previous release (see Fig. 1, 6,-7, 9, 17, 20-24) whereby derived objects are enlisted for a session of developer's view based on version control system, the system routines for matching version name and attributes per file

stored in the VOB or check-out control system are intrinsic to the above user's actions col. 20, line 45 to col. 21, lines 63), the matching of name and attributes to see if versions stored correspond to a particular release is strongly suggested. It would have been obvious for one skill in the art at the time the invention was made to implement the checkout and user's version verifying in the tree-based integration of proper files so that for each build in said candidate list as incrementally added to the tree, so that name of build (see Fig. 17) as well as attributes (see Fig. 9; Fig. 23) are also checked and verified (see audit entries – Fig. 23) to assess whether such version of build components (as candidate list) would still be appropriate to be included in the tree and enlistment view for the bug fix project (see col. 13, line 41 to col. 14, line 47; col. 9, line 8 to col. 10, line 35) .

As per claim 12, Leblang discloses further including:

for each build included in said candidate list, determining a number of matches for software modules included in said each build having a matching module name and attributes of a software module included in said first build (see rationale as set forth in claim 11 – Note: each set of files or build component per check out reads on list of candidate builds);

determining a valid list of one or more matching builds, each of said builds included in said valid list having a full match for software modules included in said each build with software modules included in said first build, each software module included in said each build having a corresponding matching software module included in said first build, said name and associated attributes of said each software module matching said matching software module included in said first build (see Fig. 23-24; col. 20, line 45 to col. 21, lines 63 – Note: the result of assessing the entries based on references from the version checking system and the configuration record

leading to derived objects reads on valid list as a result from checking on checkout builds or candidate list);

But Leblang does not explicitly disclose

determining a maybe list of one or more builds, each of said one or more builds included in said maybe list having at least one software module having a module name that does not have a matching software module included in said first build, said each build including at least one software module

having a module name and associated attributes matching another software module included in said first

build, wherein said each build has an associated number of matches.

But based on the candidate nature of the checkout build components being enlisted and checked as set forth above in claim 11, the above name and attribute mapping and determining steps would have been obvious as set forth in the rationale of claim 11.

As per claim 13, Leblang discloses

if there is only one build included in said valid list, determining that said one build matches said first build; and if there is more than one build included in said valid list, selecting one of the more than one builds included in said valid list as being the build matching said first build (see claim 11 - Fig. 23-24; col. 13, line 41 to col. 14, line 47); performing an alternative action if said valid list is empty (Note: when a release is targeted for a rebuild, new fix associated with an starting empty list reads on alternative action if the fix release for target is not founded on any legacy of record -- see col. 10, lines 59-61).

However, Leblang does not explicitly disclose if said valid list is empty, for each project, adding a build from said maybe list to said valid list in which the build added has a maximum number of matches of builds associated with said each project, a project having one or more associated builds. But the maybe list has been addressed above in claim 12, hence would have been obvious herein in regard to the rationale set forth therein.

As per claims 14-15, Leblang discloses selecting one or more build associated with a software project (e.g. version selector – col. 2, line 54 to col. 3, line 19); determining a matching build in accordance with a predetermined build selected from a list of more than one build previously registered (e.g. Fig. 9, 17, 20).

As per claims 32-36, these claims correspond to the subject matter of claims 11-15, respectively; hence will be rejected according to the rationale set forth therein, respectively.

Response to Arguments

7. Applicant's arguments filed 4/23/07 have been fully considered but they are not persuasive. Following are the Examiner's observation in regard thereto.

USC 35 § 102(b) Rejection:

(A) Applicants have submitted (Appl. Rmrks, middle para - pg. 34) Leblang's makefiles used to determine configuration record cannot establish 'extracting build information by processing modules ... compilation process resulting in the software modules', i.e. Leblang being silent about extracting information about binaries and the likes. The rejection has pointed out to portions where the source code and audit process derive objects from the VOB whereby some object files (.obj, .o files) are identified, and in such database information being retrieved additional information about those files are included as configuration data with test data, support

file entries, and pathnames. It is deemed that Leblang retrieves information regarding compilation produced files. The argument is not persuasive.

(B) Applicants have submitted that (Appl. Rmrks, bottom pg. 34, top para - pg. 35) that Leblang is silent regarding software module information being tested during execution. For lack of specifics as to the nature of the *execution*, the *test* and the *module information* being recited, broad interpretation has been applied. And the cited parts of Leblang describing a developer session attempting a developer's process to correct a software version, to debug a component or to audit on validity of store code; and those processes do read on test or execution. The Rejection has also mapped what is considered module information gathered from those developer's process, and what is execution or runtime. Applicant's arguments fail to comply with 37 CFR 1.111(b) because they amount to a general allegation that the claims define a patentable invention without specifically pointing out how the language of the claims patentably distinguishes them from the references.

(C) Applicants have submitted that Leblang does not teach or suggest 'calculating, in response to said query, said code volatility metric ... first build and said second build' (Appl. Rmrks, pg 35, bottom para). It is believed that based on the interpretation of what 'metric' represents, the cited portions of Leblang have been deemed sufficient to provide what a metric is all about. The quantity of code (a delta: symbol for a metric) changes reflected via repeated record of delta collection, each collection of such representing a measure about what is recited as 'amount of code change ... occurred between software modules...'. For lack of specifics as to the nature of the *metric* being recited, broad interpretation has been applied. Applicant's arguments fail to comply with 37 CFR 1.111(b) because they amount to a general allegation that

the claims define a patentable invention without specifically pointing out how the language of the claims patentably distinguishes them from the references.

In all along with the rejection of 35 USC 103(a) for which there are no valid arguments, the claims will stand rejected as set forth in the Office Action.

Interview Summary

8. The Applicant's Representative (att. Anne Saturnelli) was approached in a telephonic contact as per 7/6/07 to be communicated some suggestions from the Examiners concerning some subject matter that might have some weight towards improving allowability assessment if included in the independent claims. That is, the limitations such as: how the dynamics of the DB query call (if provided in the main claims) would support the utilization of compiled products and their previous DB registered module information? how determining what is appropriate for a software build established at the onset can be put forth in a clearer sequence utilizing the support derived from compiled modules and executed modules? all of which in a more understandable timeframe (e.g. delimiting when compiling ends, when runtime starts, when Db communication happens); matching with proper builds for a target program using some candidate list.

But after analysis of true nature the claimed invention, NO true agreement was reached and the Examiner indicated that it would be better that a subsequent Office Action would give the Applicant an opportunity to see how the claims can be changed based on the observations made by the Examiner in such action.

Conclusion

9. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Tuan A Vu whose telephone number is (272) 272-3735. The examiner can normally be reached on 8AM-4:30PM/Mon-Fri.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Meng-Ai An can be reached on (571)272-3756.

The fax phone number for the organization where this application or proceeding is assigned is (571) 273-3735 (for non-official correspondence - please consult Examiner before using) or 571-273-8300 (for official correspondence) or redirected to customer service at 571-272-3609.

Any inquiry of a general nature or relating to the status of this application should be directed to the TC 2100 Group receptionist: 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished

applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Tuan A Vu
Patent Examiner,
Art Unit 2193
July 14, 2007